Peer-Review 1: UML

Gruppo **GC6** - Rigamonti Alberto, Rogora Matteo, Sarbu Razvan Irinel

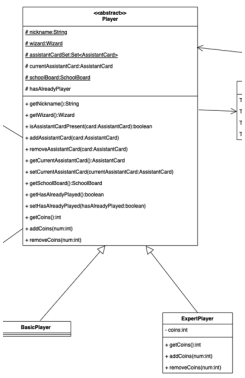
Valutazione del diagramma UML delle classi del gruppo **GC16**.

Lati positivi

Abbiamo riscontrato una serie di lati positivi durante l'analisi dell'UML del gruppo GC16. In particolare abbiamo apprezzato la selezione della modalità di gioco tramite estensione come per le classi *Game* e *ExpertGame*.
Un ulteriore lato positivo è la modalità di gestione delle isole e raggruppamenti di isole, che funziona correttamente senza usare un numero eccessivo di metodi.
È stata inoltre apprezzata la disposizione degli elementi del modello in mappe e liste, che risultano più gestibili di implementazioni che sfruttano array di interi.
Simulando l'esecuzione di una partita non abbiamo riscontrato problemi nell'utilizzo dei metodi proposti, ad eccezione della gestione dei Character in expert mode. Complessivamente risulta giocabile senza troppe difficoltà.
Infine la nomenclatura di metodi e attributi risulta intuitiva e autoesplicativa e molto spesso non necessita di ulteriori informazioni per comprendere la funzione da essi svolta.

Lati negativi

L'utilizzo del solo colore per studente è stato punto di dibattito nel nostro gruppo, può essere considerato sia un vantaggio poiché alleggerisce la struttura del modello, ma anche uno svantaggio nel caso in cui sia necessario selezionare uno studente in particolare tra più studenti dello stesso colore. Il passaggio di soli colori potrebbe causare difficoltà di implementazione nella View.



Per la classe Player è possibile rimuovere <<abstract>> dato che la sottoclasse BasicPlayer è vuota. Per utilizzare un player in una partita "Normal" basterà istanziare un oggetto di tipo Player. La classe BasicPlayer verrà dunque rimossa.
Lo stesso procedimento è applicabile per la classe Game e BasicGame.

Di conseguenza, per entrambe le classi non è necessario definire i metodi della “ExpertMode” all’interno della rispettiva classe “NormalMode”, in quanto già presenti nella sottoclasse.
All’interno della classe ExpertGame abbiamo considerato il metodo addCharacterCard() superfluo, dato che i characters non sono selezionati dai Player ma estratti, dunque è possibile istanziare le CharacterCard necessarie già nel costruttore.
La classe Controller risulta molto pesante, a nostro parere sarebbe meglio dividerla in più sottoclassi che gestiscono le varie fasi del gioco.
Gli effetti dei singoli personaggi andrebbero gestiti nel Controller, poiché sono variabili e dipendenti dalla chiamata del client (presentano della logica di selezione, come ad esempio per scegliere quali studenti posizionare su specifiche “tiles” di gioco nel caso del Monaco). Anche il calcolo dell’influenza andrebbe effettuato nel Controller, poiché dipendente da molti effetti dei personaggi in expert-mode (all’interno della classe IslandGroup basterebbe il metodo che ritorna il numero di studenti di un determinato colore).

Confronto tra le architetture

Un elemento di distinzione delle nostre architetture, dal quale potremmo prendere spunto, è la selezione della fase del turno come stato e, di conseguenza, il salvataggio dello stesso all’interno di game o di una classe apposita nel modello.
I nostri modelli risultano complessivamente simili, dunque risulta difficile trovare altri elementi differenti, fatta eccezione per alcune scelte implementative riguardo le quali abbiamo deciso di continuare ad utilizzare la nostra versione, come per il collegamento tra game e controller, poiché vogliamo implementare la funzionalità di multiple partite. Nel nostro caso, il Controller presenta una mappa contenente multiple istanze di gioco riconosciute univocamente da un codice e ciò porta alla completa separazione della logica di gioco e di tutti gli attributi relativi allo stato del Modello, obiettivo condiviso inizialmente anche dal gruppo revisionato.